Scientific Libraries for Cray Systems: Current Features and Future Plans

Mary Beth Hribar, *Cray Inc.*, **Chip Freitag**, *AMD*,
Adrian Tate, *Cray Inc.*, **and Bracy Elton**, *Cray Inc.*

ABSTRACT: *Cray continues to provide support and performance for the basic scientific computations that Cray's customers require. On the vector-based systems, LibSci is the scientific library package provided by Cray. On the Cray XD1 and the Cray XT3 systems, Cray distributes AMD's Core Math Library (ACML) and a much smaller Opteron LibSci package. This talk will describe the current and future scientific library support for these Cray systems. In addition, performance results will be given for LibSci and ACML.*

KEYWORDS: LibSci, ACML, Cray X1, Cray X1E, Cray XD1, Cray XT3, scientific libraries

1. Introduction

Scientific libraries are a component of Cray's programming environment software. For each Cray system, these libraries provide basic numerical functions that have been highly tuned for that architecture. To access the best performance of Cray systems, use these libraries.

For the Cray XD1™ and Cray XT3™ systems, the AMD Core Math Library (ACML) provides most of the scientific library support. For convenience, ACML is included in Cray's software distribution. On the Cray X1™ series systems, LibSci® contains the scientific library routines.

This paper states the features, release schedules and plans for Cray's scientific libraries and ACML.

2. Libraries for Cray systems

Currently, Cray offers three different products. The software differs between these products, including the scientific libraries. The Cray X1 and Cray X1E systems contain vector processors. Cray develops the operating system, compilers and libraries for these systems. The Cray XT3 and Cray XD1 contain AMD Opteron™ processors. The software for these systems is a combination of third party software and libraries developed by Cray.

The features of the scientific libraries vary between these three Cray products. It is a goal to make the libraries more consistent across all Cray platforms in the future.

2.1 Cray X1 series

The Cray X1 system and its upgrade, the Cray X1E system, combine vector processors with both shared and distributed memory. These systems are constructed of nodes within a node interconnection network, each of which contains four multistreaming processors (MSPs) and globally addressable shared memory. Each MSP contains four single-streaming processors (SSPs).

The scientific libraries for the Cray X1 series are contained in LibSci. LibSci provides Fortran interfaces for all routines and support for MSP mode, SSP mode, and 32- and 64-bit data types. The latest release is LibSci 5.4.

2.1.1 Single processor routines

LibSci contains single processor support for:

- Fast Fourier transform (FFT), convolution, and filtering routines
- Sparse direct solvers
- Basic Linear Algebra Subprograms (BLAS)
- Linear Algebra Package (LAPACK) routines

2.1.2 Distributed memory parallel routines

LibSci contains multiprocessor support in a distributed memory environment for:

- FFT routines
- Scalable LAPACK (ScaLAPACK) routines
- Basic Linear Algebra Communication Subprograms (BLACS)

2.1.3 Shared memory parallel routines

LibSci also contains four-way shared memory parallel support across a single node for all Level 3 BLAS routines and for the Level 2 BLAS routines `sgemv`, `dgemv`, `cgemv`, and `zgemv`. This library is implemented with OpenMP, and including the `-lompsci` option on the link line accesses it.

2.1.4 Inlining LibSci

There is a small set of LibSci routines that can be inlined with the `–O inlinelib` option. All Level 1 BLAS routines can be inlined as well as some Level 2 BLAS routines (`sgemv`, `dgemv`, `cgemv`, `zgemv`, `sger`, `dger`, `cgerc`, `cgeru`, `zgerc`, and `zgeru`).

2.2 Cray XT3

The Cray XT3 is a massively parallel processing (MPP) system designed to provide the scalability, performance and reliability to solve the most demanding high performance computing problems. Cray XT3 systems are based on Red Storm, the system developed jointly by Cray Inc. and the U.S. Department of Energy's Sandia National Laboratories.

The Cray XT3 contains single processor nodes in a high-bandwidth, low-latency 3-D torus interconnect. Each node includes an AMD Opteron processor, a dedicated memory and a Cray SeaStar™ communication chip. The Cray SeaStar chip contains a HyperTransport™ link to the Opteron processor, a Direct Memory Access (DMA) engine to manage memory accesses, and a router that connects to the system interconnection network. The Cray SeaStar chip offloads the communications processing from the Opteron processor, increasing the efficiency of the computation and communication within an application program.

There are two types of nodes in the system. Service nodes perform the functions needed to support users, administrators, and applications running on compute nodes. These nodes run a full-featured version of SuSE LINUX. Compute nodes run a microkernel named *Catamount*. Catamount was developed by Sandia National Laboratories to provide support for application execution without the overhead of a full operating system image. Programming environment software, including scientific libraries, is provided on both types of nodes.

The Cray XT3 programming environment includes versions of the 64-bit AMD Core Math Library (ACML) to support the GNU and PGI compilers. The compiler drivers automatically load and link to the PGI compatible `libacml` when the `PrgEnv` module is loaded. It is not necessary to load and link manually as described in the ACML documentation. If a user wishes to link to the 64-bit GNU compatible library, the user must swap the `acml` module with the `acml-gnu` module.

The Cray XT3 programming environment also includes a scientific libraries package, Cray XT3 LibSci. Note that the Cray XT3 LibSci differs from the LibSci for the Cray X1 series systems. It contains a much smaller set of library routines. The Cray XT3 1.1 release of Cray XT3 LibSci will include:

- ScaLAPACK
- BLACS
- SuperLU_DIST

More features are planned for Cray XT3 LibSci. In future releases, the Cray interface for serial and distributed memory parallel FFTs will be provided.

2.3 Cray XD1

The Cray XD1 is also an MPP system comprised of AMD Opteron processors. The Cray XD1 introduces a new architecture, the Direct Connected Processors, which employs the RapidArray™ interconnect to provide a high-bandwidth, low-latency network. The RapidArray processors connect the Opteron processors to the RapidArray switching fabric. The RapidArray processors offload the communication functions from the Opteron and accelerate them in hardware.

The Cray XD1 operating system is based on the SuSE Linux Enterprise Server (SLES) distribution. This operating system supports both 32- and 64-bit applications. The programming environment software includes 32- and 64-bit ACML libraries, with versions to support the GNU and PGI compilers. The OpenMP ACML is available in 32- and 64- bit versions for use with the PGI compilers. ScaLAPACK and BLACS are also provided.

In the 1.1 and 1.2 Cray XD1 software releases, the module environment does not include modules for ACML, ScaLAPACK or BLACS. Please link to ACML according to the guidelines in the *AMD's Core Math Library User's Guide*, or request that your system administrator install modules for ACML. The ScaLAPACK and BLACS libraries are located in the `/usr/local/lib64` directory.

3. ACML

The AMD Core Math Library (ACML) is a package of numerical routines tuned specifically for the AMD64 platform processors, including the Opteron. This library provides the basic numerical functions for the Cray XT3 and Cray XD1 systems. ACML contains:

- BLAS
- Sparse Level 1 BLAS
- LAPACK
- ACML FFTs

Also, pre-built ScaLAPACK and BLACS libraries are included with ACML. Since Cray distributes tuned ScaLAPACK and BLACS libraries, the ACML versions of these libraries are not included in Cray's software distributions.

3.1 Fortran and C interfaces to ACML

In addition to the standard Fortran interface to the library routines, there are C interfaces to the routines in ACML. To call ACML routines from a C program, the user must include the header file `acml.h`. This file contains prototypes for all C interfaces, as well as prototypes of the Fortran interfaces. The C programmer has the choice of calling either the C or the Fortran interface to the ACML routines.

The C interface differs from the Fortran interface. To distinguish the Fortran routines from the C routines, the Fortran routine names are in uppercase and appended by an underscore. The C interfaces do not include any workspace arguments. All workspace is allocated locally within the routines. The scalar arguments in the C interfaces are passed by value. There is no native complex data type in C. Routines that operate on complex data use the data types `complex` and `doublecomplex` defined in `acml.h`.

Both the Fortran and the C interfaces require that two-dimensional arrays be stored in column major order. This is the native order for two-dimensional arrays in Fortran, but not in C.

More details about calling ACML from C programs are included in the *AMD's Core Math Library User's Guide*.

3.2 BLAS and LAPACK in ACML

The ACML library contains level 1, level 2 and level 3 BLAS routines, which provide the basic vector and matrix computations. These are highly tuned for the Opteron processor. Sparse level 1 BLAS routines are also provided.

ACML also contains LAPACK, the dense linear algebra package. Most of the performance of LAPACK routines benefit from the highly tuned BLAS routines and are written using block algorithms wherever possible. The ACML LAPACK routines contain further optimizations so they differ from the public domain LAPACK source but maintain the same level of accuracy

Level 2 and level 3 BLAS routines and some LAPACK routines are available in the OpenMP version of ACML.

3.3 ACML FFT routines

ACML provides a set of highly tuned FFT routines with an interface to those routines that is unique to ACML. Since there is no established standard for FFTs, this interface is different from the Cray FFT interface. Codes that call the Cray FFTs will need to rewrite these FFT calls to match the ACML FFT interface.

For full documentation of the ACML FFT routines, refer to the *AMD Core Math Library User's Guide*. A few highlights are given here.

There are routines to compute one-dimensional, two-dimensional and three-dimensional complex-to-complex FFTs. There is also a routine to compute multiple complex-to-complex one-dimensional FFTs. The standard versions compute in place with unit stride and fixed scale. The expert versions of the routines (routine names are appended with an "X") allow for out-of-place computation with selectable scales. The one-dimensional and two-dimensional expert FFT interfaces also allow a non-unit stride. The three-dimensional expert FFT interface only allows a unit stride.

There are also routines to compute one-dimensional real-to-complex and complex-to-real FFTs (single and multiple). These routines compute in place with a unit stride and fixed scale. Also, the complex data is stored in an unusual manner that is documented in the user's guide. Make a note of this format when using complex results from the real-to-complex routines or inputting complex data into the complex-to-real routines.

There are OpenMP versions of the two-dimensional and three-dimensional complex-to-complex FFT routines in ACML, in the `acml-mp` version. There are no distributed memory parallel FFT routines in ACML.

3.4 Fast and vector math functions in ACML_MV

Optimized libm is available in glibc in the SuSE SLES9 and SL9.x distributions. Using the fast math routines in ACML_MV can provide further performance improvements. These fast routines may sacrifice accuracy, so the user should determine if the arguments are suitable to use the fast math routines.

The ACML_MV library also contains vector intrinsics of some of the libm routines. These intrinsics can be called in assembly language, or by C compilers that support XMM register m128 data types (such as gcc). The PGI compilers do incorporate these vector intrinsics when producing optimized code. Finally, the ACML_MV library contains array versions of the libm routines. These are callable from C or Fortran, and provide an efficient way to perform the desired transcendental function on an array of *n* input values. The currently supported libm functions in ACML_MV are log, log10, logf, exp, expf, sin, cos, sincos, pow (scalar only), and powf (scalar and vector).

3.5 Upcoming features

A release of ACML is scheduled at the end of June to coincide with ISC 2005. This release will provide a comprehensive random number generation suite. It will also provide an FFT plan builder utility to provide more optimal performance for non power-of-2 FFT sizes.

Further work is also planned for ACML_MV. The next release will include sinf, cosf, and sincosf. More routines are planned for the future including the inverse and hyperbolic trigonometric functions.

New releases of ACML are planned for approximately every 6 months.

4. LibSci enhancements

Cray is providing an internally developed set of numerical routines on each Cray platform. For the distributed memory parallel routines, it is important to tune them for the specific interconnect and message passing libraries of Cray's systems. Also, Cray supports its own interface for FFT routines and we plan to offer this interface on all platforms. These routines must then be tuned for each Cray system also. Finally, we plan to add more support for sparse solvers.

4.1 ScaLAPACK and BLACS tuning

At Cray, we are developing techniques to improve performance of ScaLAPACK on all systems. Most of the work will apply to all systems, but there are optimizations that are platform specific. Our ScaLAPACK tuning is partly joint work with the ScaLAPACK team, namely Osni Marques and Tony Drummond at NERSC.

4.1.1 General algorithmic improvements to ScaLAPACK

We are investigating ways to improve the efficiency of the solvers in ScaLAPACK. Our approach is to rewrite certain parallel algorithms to remove redundancies and to overlap computational phases with communication phases. As part of this work, new communication routines will be added to the BLACS.

We decided to start with one of the most heavily used ScaLAPACK routines, `pzgetrf`, which computes the LU factorization of a matrix. Our aim is to improve the concurrency of the required process of exchanging matrix rows for numerical stability. To support such changes to the LU factorization algorithm, we must have the ability to address memory or query for a result on another processor with that remote processor being only partially involved. That partial involvement, and the removal of the need to handshake between processors is key to making the modified algorithm perform more efficiently than the original.

We defined a new BLACS routine, `blacs_spin`, which causes a processor to spin until some condition has been satisfied on a remote processor. The remote processor is not explicitly involved in this interaction, and can continue executing its program without being affected. This is tremendously useful when the algorithm contains some kind of inherent serial sections or redundancy. It is now possible for previously idle processors to query for results and to proceed independently once those results are obtained.

We now examine the steps of the original algorithm and the modified algorithm. Both algorithms advance the factorization in block steps, each adding one block column (panel) and block row to the factorization. The original routine `pzgetrf` proceeds as follows:

1. Compute the factor entries for the current panel. This proceeds column by column and requires row interchanges in which the current diagonal entry is replaced by the largest entry in the same column. The swap or pivot information is saved.
2. Broadcast the pivot information to all process columns
3. All processors perform the pivoting row interchanges, possibly requiring interchanges with other processors
4. Broadcast the entries of the diagonal block of the LU factorization to all process columns.
5. All processors modify their data for later block columns accordingly.

Using the new BLACS routine we can allow steps 2 and 3 to proceed concurrently with step 1. Processors in other process columns check for new pivot information on the process that contains it, without the latter processor being involved. That is, while processors in one process column execute step 1, processors without block panel data check for pivot updates using `blacs_spin`. When `blacs_spin` returns a pivot, the processor performs a row swap.

Our revised scheme has better concurrency, but increases the number of messages while retaining the same volume of remote memory access. We can make a further improvement by introducing a threshold pivoting scheme commonly used in sparse factorization methods. Rather than exchanging the diagonal entry with the largest column entry, we exchange with some suitably large entry. If any suitably large entry is found on the processor that holds the diagonal, we perform the interchange locally, thereby reducing interprocessor row swaps. "Suitably large" is defined by a user-specified threshold, allowing the user to make a compromise between efficiency and numerical stability appropriate for their application. This new scheme is being developed currently for the Cray XT3 and Cray X1 series systems.

Initial performance results indicate that the modified algorithm performs better than the original on the Cray X1 series systems. As the problem size increase, the improvement increases. More tests are needed with threshold pivoting, and with larger problems and processor grids.

The parallel LU factorization routines, psgetrf, pdgetrf, pcgetrf, pzgetrf, will contain performance improvements in the next software release for the Cray X1 series, scheduled for December 2005.

4.1.2 Platform specific improvements

In addition, we optimize for each of the architectures that we support. This includes:

- Cray XD1: decoupling block size from distribution block size, tuning BLACS for the RapidArray interconnect

- Cray X1 series: using Co-Array Fortran in communication intensive areas
- Cray XT3: decoupling block size and tuning for the Cray XT3 MPI implementation, using shmem low-latency message passing

4.2 Cray FFT development

The Cray FFTs continue to be tuned for the Cray X1 series, and work has begun to port the Cray FFT interface to the Cray XD1 and Cray XT3 systems.

4.2.1 New features for Cray X1 series

Many new features and performance enhancements have been added to the FFTs in the last three releases of the Programming Environment (PE) software.

In the PE 5.2 software release (April 2004), distributed memory parallel FFT routines were added to LibSci. The interface for these parallel routines is consistent with the parallel FFT routines offered on Cray T3E systems.

In the PE 5.3 software release (December 2004), the distributed memory parallel FFT routines were modified to accept additional work array arguments. This change provides the option to manage memory that would otherwise be handled internally. Also, special complex-to-complex butterflies were added for some composite radices, e.g., 6, 10, 12, 15, and 20, and higher powers of 2, e.g., 16. Together, they reduce the number of floating-point and memory operations, thus generally improving the performance of the FFTs. Improvements are seen in the complex-to-complex routines as well as the complex-to-complex phase of multidimensional real-to-complex or complex-to-real transforms. Furthermore, performance of multiple 1-D and multidimensional FFTs is generally improved with more effective use of the cache.

In the PE 5.4 software release (March 2005), complex-to-complex FFT butterflies for radices 7, 11, and 13 were implemented. Applications that previously involved complex-to-complex FFTs with FFT lengths containing products of powers of these radices will run faster than with previous releases. This also includes the convolution routine CCNVLF, which effects a complex convolution via the Fourier transform.

4.2.2 Performance on Cray X1 series

Many performance enhancements to the FFTs have been introduced in LibSci between the PE 5.2 and PE 5.4 software releases. Included here are graphs of selected complex-to-complex FFT routines, comparing the performance of PE 5.2 to PE 5.4. The ratio of time to compute FFTs for PE 5.4 to the time for PE 5.2 is plotted against problem size. A value below one indicates a performance improvement. These results are for single precision routines with 64-bit default data types, using leading dimensions yielding strides that are odd multiples of four.

The tests were run on a single MSP of a Cray X1 system using Cray LibSci FFTs in PE 5.2 and in PE 5.4. The ISYS parameter was 0 in all cases, and the FFT lengths shown contain only factors that are powers of 2, 3, and 5. Figures 1-4 depict an overall improvement with the most improvement for three-dimensional FFTs. The plots are somewhat jagged as some FFT lengths take advantage of the new butterflies more than other lengths.

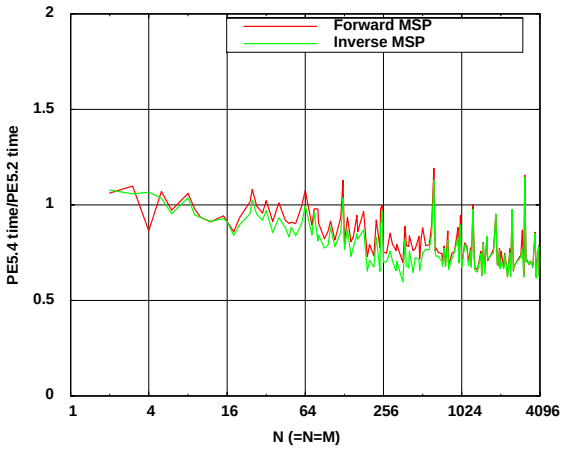

Figure 1. Improvement of 64-bit CCFFTM (MSP mode) on Cray X1, given by ratio of PE 5.4 time to PE 5.2 time.

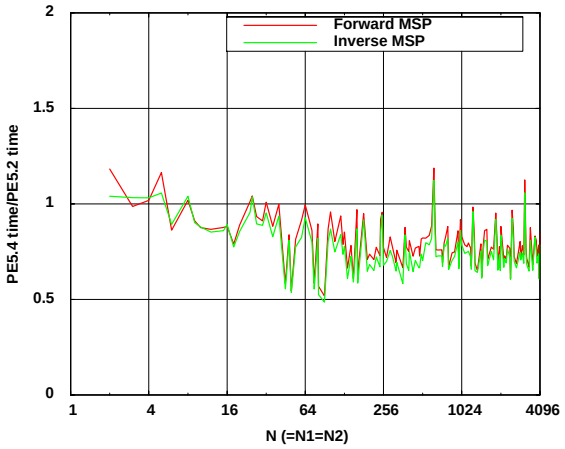

Figure 2. Improvement of 64-bit CCFFT2D (MSP mode) on Cray X1, given by ratio of PE 5.4 time to PE 5.2 time.

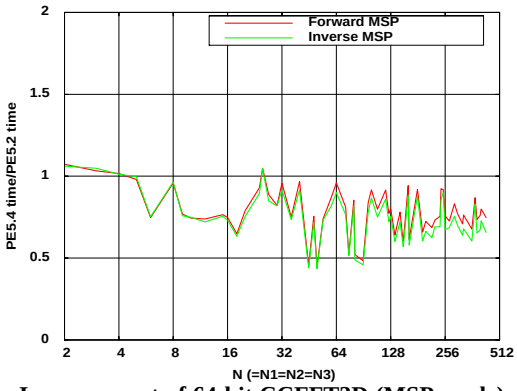

Figure 3. Improvement of 64-bit CCFFT3D (MSP mode) on Cray X1, given by ratio of PE 5.4 time to PE 5.2 time.

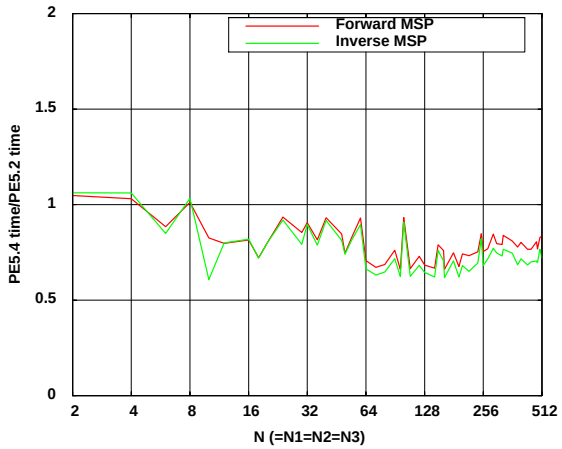

Figure 4. Improvement of 64-bit SCFFT3D (MSP mode) on Cray X1, given by ratio of PE 5.4 time to PE 5.2 time.

4.2.3 Cray FFT interface on Cray XT3 and Cray XD1

We plan to provide the Cray FFT interfaces on the Cray XT3 and Cray XD1 systems in future software releases. It is our goal to have a consistent FFT interface across all Cray platforms. And, we wish to add to the FFT functionality that is provided in ACML. The ACML FFTs do not provide a complete set of real-to-complex, complex-to-real FFTs or distributed memory parallel routines. And, the ACML complex 3-D FFT routine CFFT3D does not allow leading dimension parameters.

The initial port of the Cray FFT interfaces to the Opteron processor is complete. It is our intent to incorporate ACML FFTs in the Cray FFTs where it is possible and makes sense to do so. We have yet to determine whether ACML implementations will replace or augment Cray FFT implementations. In either case it is necessary to meet with AMD and Numerical Algorithms Group (NAG) to learn more about the ACML FFT implementation, and to suggest further features to extend the applicability of the ACML FFTs to Cray FFTs.

The amount of work required to provide highly tuned performance of the Cray FFTs on Cray XT3 and Cray XD1 systems is still being determined.

4.3 Sparse solvers

Cray plans to make SuperLU our standard direct linear equation solver on all platforms. We are in discussions with NERSC to develop a common interface for the presently three separate machine model versions of SuperLU.

Optimized sparse level 1 BLAS are provided on all Cray platforms, either as part of LibSci or in ACML. We plan to support optimized sparse matrix-vector multiplication routines on all platforms, in ways that enable users of PETSc or Trilinos to take full advantage of Cray hardware.

Supporting sparse eigensolutions through ARPACK / PARPACK is possible, if we learn that this would be important to our users.

5. Summary

Cray offers a set of scientific library routines on each Cray platform. Our goal is to make these libraries more consistent between systems. Features planned for future Cray XD1 and Cray XT3 software releases will provide an FFT interface and a library module environment that are consistent with other Cray systems.

AMD's Core Math Library (ACML) provides highly tuned scientific libraries for Opteron processors, and it is included in the Cray XD1 and Cray XT3 software distributions. Cray and AMD are working together to ensure that ACML contains the performance and features required by Cray's customers.

Cray continues to improve algorithms and implementations for their scientific libraries. Collaborations with universities and government labs provide assistance with this work, and more joint projects are encouraged.

Acknowledgments

The authors would like to acknowledge all of those who contributed to this paper and to thank AMD for their support in providing a presentation at CUG. Tim Wilkens at AMD provided performance information for ACML. We thank Preeta Raman at AMD for coordinating discussions between Cray and AMD. We acknowledge the following members of the Cray Scientific Libraries Group: Chao Yang who optimizes the BLAS, LAPACK, sparse solvers and the linpack benchmark, Neal Gaarder who tunes libm, and John Lewis who provides technical assistance in all areas of linear algebra and who develops sparse routines. We also acknowledge Ursula Kallio, the technical writer for Cray's scientific libraries and Catherine Knutson, the integrator and tester of Cray's libraries.

About the Authors

Mary Beth Hribar is the manager of the Cray Scientific Libraries Group. She can be reached at Cray

Inc., 411 First Ave S, Suite 600, Seattle WA 98104. Her email address is marybeth@cray.com.

Chip Freitag is the project manager for the math libraries at AMD. He can be reached at AMD, 5204 E. Ben White Blvd, MS 621, Austin TX 78741. His email address is chip.freitag@amd.com.

Adrian Tate is a member of the Cray Scientific Libraries Group. He is the project lead for ScaLAPACK tuning. He can be reached at Cray Inc., 411 First Ave S, Suite 600, Seattle WA 98104. His email address is adrian@cray.com.

Bracy Elton is a member of the Cray Scientific Libraries Group. He is the project lead for FFT development and tuning. He can be reached at Cray Inc., 411 First Ave S, Suite 600, Seattle WA 98104. His email address is elton@cray.com.